Abstraction for Real-time Intelligent Control: Extended Abstract

Robert P. Goldman, David J. Musliner, Mark S. Boddy, Kurt D. Krebsbach
Automated Reasoning Group
Honeywell Technology Center
3660 Technology Drive
Minneapolis, MN 55418
{goldman,musliner,boddy,krebsbac}@htc.honeywell.com

Introduction

In this paper we discuss two abstraction techniques we use in CIRCA planning (Musliner, Durfee, & Shin 1993; 1995). CIRCA agents construct and execute plans for controlling real-time systems that interact with a dynamic environment including uncontrolled, exogenous events. In these environments, catastrophic failure is possible if a *timely* control action is not taken in certain situations. Control plans for these environments must provide guarantees that failures will not occur. Therefore, the CIRCA planning problem is one of generating a timed, discrete event controller (Ostroff & Wonham 1990) that attempts to achieve goals while delivering performance guarantees. In this generation process we abstract temporal information, using a special-purpose temporal prover to summarize information about the latency of various events. We also use a technique we call Dynamic Abstraction Planning (Goldman *et al.* 1997) to minimize the (non-temporal) feature space of the controllers.

In this position paper we present the two abstraction methods used in CIRCA state-space planning. We start by reviewing the CIRCA architecture, which couples a deliberative planning component with a scheduler and a real-time executive. We then present the state-space planning problem, which is the responsibility of CIRCA's AI Subsystem. We then discuss temporal abstraction in state-space planning, and then present Dynamic Abstraction Planning (DAP), a planning technique that dynamically and *locally* abstracts the feature space of the controller NFA. We relate these techniques to methods used in AI planning, Model (automaton) Minimization and Markov Decision Processes.

CIRCA

CIRCA is designed to support both hard real-time response guarantees and unrestricted AI methods that can guide those real-time responses. Figure 1 illustrates the architecture, in which an AI subsystem (AIS) reasons about high-level problems that require its powerful but potentially unbounded planning methods, while a separate real-time subsystem (RTS) reactively executes the AIS-generated plans and enforces guaranteed response times. The AIS and Scheduler modules cooperate to develop executable reaction plans that will assure system safety and attempt to achieve system goals when interpreted by the RTS.

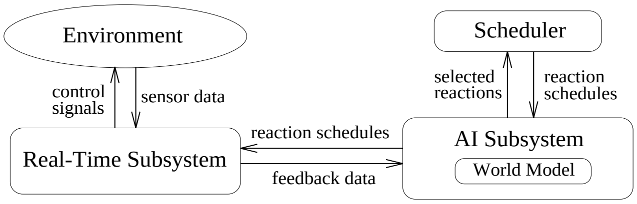


Figure 1: The Cooperative Intelligent Real-Time Control Architecture.

CIRCA has been applied to real-time planning and control problems in several domains including mobile robotics and simulated autonomous aircraft. In this paper we draw examples from a domain in which CIRCA controls a simulated Puma robot arm that must pack parts arriving on a conveyor belt into a nearby box. The parts can have several shapes (e.g., square, rectangle, triangle), each of which requires a different packing strategy. The control system may not initially know how to pack all of the possible types of parts— it may have to perform some computation to derive an appropriate box-packing strategy. The robot arm is also responsible for reacting to an emergency alert light. If the light goes on, the system must push the button next to the light before a fixed deadline.

In this domain, CIRCA's planning and execution subsystems operate in parallel. The AIS reasons about an internal model of the world and dynamically programs the RTS with a planned set of reactions. While the RTS is executing those reactions, ensuring that the system avoids failure, the AIS is able to continue executing heuristic planning methods to find the next appropriate set of reactions. For example, the AIS may derive a new box-packing algorithm that can handle a new type of arriving part. The derivation of this new algorithm does not need to meet a hard deadline, because the reactions concurrently executing on the RTS will continue handling all arriving parts, just stacking unfamiliar ones on a nearby table temporarily. When the new box-packing algorithm has been developed and integrated with additional reactions that prevent failure, the new schedule of reactions can be downloaded to the RTS.

```
EVENT emergency-alert                          ;; Emergency light goes on
      PRECONDS: ((emergency nil))
      POSTCONDS: ((emergency T))

TEMPORAL emergency-failure                     ;; Fail if don't attend to
      PRECONDS: ((emergency T))                ;; light by deadline
      POSTCONDS: ((failure T))
      MIN-DELAY: 30 [seconds]

ACTION push-emergency-button
      PRECONDS: ((part-in-gripper nil))
      POSTCONDS: ((emergency nil) (robot-position over-button))
      WORST-CASE-EXEC-TIME: 2.0 [seconds]
```

Figure 2: Example transition descriptions given to CIRCA's planner.

CIRCA's planning system builds reaction plans based on a world model and a set of formally-defined safety conditions that must be satisfied by feasible plans (Musliner, Durfee, & Shin 1995). To describe a domain to CIRCA, the user inputs a set of transition descriptions that implicitly define the set of reachable states. For example, Figure 2 illustrates several transitions used in the Puma domain. These transitions are of three types:

Action transitions represent actions performed by the RTS.

Temporal transitions represent the progression of time and continuous processes.

Event transitions represent world occurrences as instantaneous state changes.

The AIS plans by generating a nondeterministic finite automaton (NFA) from these transition descriptions. The AIS assigns to each reachable state either an action transition or `no-op`. Actions are selected to *preempt* all transitions that lead to failure states, thus ensuring system safety. Actions are also selected to drive the system towards states that satisfy as many goal propositions as possible. The assignment of actions determine the topology of the NFA: preemption of temporal transitions removes edges and assignment of actions adds them.

The NFA is then translated into a memoryless controller for downloading to the RTS. This is done through a two-step process. First, the action assignments in the NFA are compiled into a set of *Test-Action Pairs*(TAPs). The tests are a set of boolean expressions that distinguish between states where a particular action is and is not to be executed. The test expression is a function of the plan as a whole, rather than local action assignments, because the same action may be assigned to more than one state.

Eventually, the TAPs will be downloaded to the RTS to be executed. The RTS will loop over the set of TAPs, checking each test expression and executing the corresponding action if the test is satisfied. The tests consist only of sensing the agent's environment, rather than checking any internal memory, so the RTS is asynchronous and memoryless.

However, before the TAPs can be downloaded, they must be assembled into a loop that will meet all of the deadlines. These deadlines are captured as constraints on the maximum period of the TAPs. This second phase of the translation process is done by the scheduler. In this phase, CIRCA's scheduler verifies that all actions in the TAP loop will be executed quickly enough to preempt the transitions the planner has determined need preempting. The tests and actions that the RTS can execute as part of its TAPs have associated worst-case execution times that are used to verify the schedule. It is possible that scheduling will not succeed. In this case, the AIS will backtrack to the planner for the NFA to be revised, a new set of TAPs generated and scheduled. The planning process is summarized in Figure 3.

CIRCA Planning Model

In this paper, we are concerned with the problem of *state-space planning* in CIRCA. This is the generation of the NFA describing the (real-time) discrete event controller. The state-space planning model for CIRCA consists of *states*, each of which has an associated assignment of values to the set of features, and *transitions*, which allow movement from state to state. Transitions can be partitioned on the basis of *volition*: in the current planner, *actions* are volitional, *events* and *temporals* are not. A transition is *enabled* in any state for which its *preconditions* are satisfied. The possible states resulting from a transition from a given state are those satisfying the transition's *postconditions*.

Transition postconditions are specified, per the conventional STRIPS assumption, by listing only those literals that change values — other literals retain their values. The STRIPS assumption is loosened to a limited extent by permitting nondeterministic actions;[1] such

[1]There is no need to have nondeterministic events or

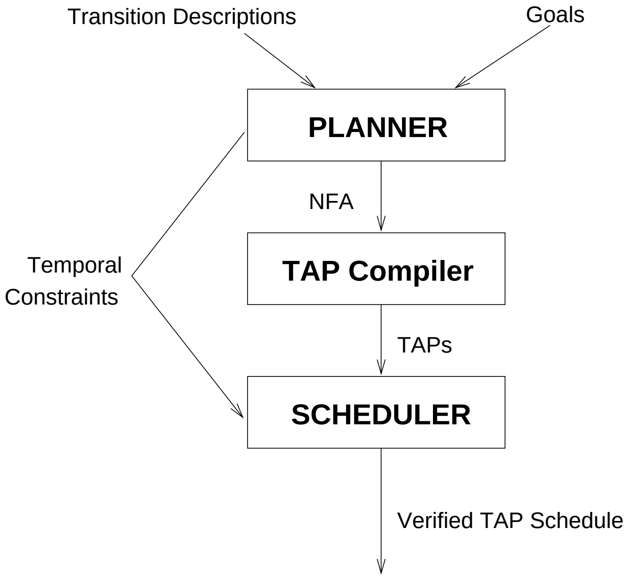


Figure 3: Summary of the CIRCA planning process.

actions have multiple sets of postconditions. The *delay* associated with a transition is known only in terms of upper and lower bounds. For a sample set of transition descriptions, see Figure 2.

In the simplest model of CIRCA planning, a CIRCA state-space *plan* is a graph, in which nodes correspond to states and arcs represent enabled transitions between these states. A transition is enabled if its preconditions are satisfied by the state features. This simple model corresponds to the original CIRCA planner (Musliner, Durfee, & Shin 1995).

A *well-formed* plan is one in which an arc is present for every enabled nonvolitional transition between nodes which is not *preempted*, and each node has outgoing arcs for at most one action (non-deterministic actions may require more than one arc).

A non-volitional transition, t may be preempted by planning an action that is guaranteed to occur before t can. That is, an action whose delay is less than the latency of t. For example, in the domain described in Figure 2, CIRCA would attempt to preempt `emergency-failure` with the action `push-emergency-button`. This may or may not succeed, depending on the context in which the preemption is attempted. It is not enough to choose an action that is faster than the delay on the temporal: the state-space planner must also ensure that this action is started before too much of the delay has elapsed.

Timing information for a CIRCA plan is derived from bounds on the delay associated with arcs out of a node, which is taken directly from the delay bounds for the corresponding transitions. The *latency* of a transition arc with respect to a node in the plan is the time before that transition will occur, if no other transition occurs first, once that node has been reached.

temporals — for those it suffices simply to have multiple transitions.

The model we have outlined is a simplified timed discrete-event control model (Ostroff & Wonham 1990). Kabanza provides a controller synthesis algorithm that works with such models in a straightforward way, interleaving non-volitional and volitional transitions, and with state feature spaces augmented by a feature corresponding to elapsed time (Kabanza, Barbeau, & St.-Denis 1997). However, in the applications we have examined, this approach is not feasible. The reason is that our applications involve both very long latency processes (e.g., the warming up of a rocket engine) and short latency processes (tests and actions in the RTS controller loop). Explicitly recording temporal indices would yield an enormous state-space explosion. Accordingly, we treat time specially.

Another complicating issue is the fact that the latency of a transition in a particular state is path-dependent. For example, consider a state A, in which a temporal transition t is enabled. The delay of transition t is d_t. Let us assume the controller can reach state A from two other states, B and C; in state B, transition t is enabled, and in state C it is not. Now, the latency of t in state A depends on how state A is reached: if it is reached from state B, then the latency is $d_t - \mathrm{dwell}(B)$[2]; if from state C, the latency is simply d_t. This is only a very simple example; in a real problem there are typically many ways to reach a given state and in general we need to deal with chains of multiple states in which a transition is enabled, not just a single predecessor. In the following section we discuss how we address the problem of temporal abstraction in CIRCA state-space planning.

Temporal Abstraction

We wish to avoid explicitly recording temporal information in the states of the CIRCA NFA. However, as mentioned above, the latency of a transition in a state depends on the path along which that state was reached, which breaks the Markov assumption for nodes. We restore this property by calculating and employing path-independent bounds on latency in providing timing guarantees (most significantly, in determining preemption of transitions by actions). This abstraction is critical to successful function of the CIRCA state-space planning algorithm, but regrettably sacrifices completeness.

To understand the technique, one must consider how the state-space planner establishes the safety of a controller. The controller is an NFA containing edges for actions and non-volitional transitions and a distinguished failure state, reached through transitions to failure (e.g., `emergency-failure` in Figure 2. A controller is safe if the distinguished failure state is not reachable from the start state(s). The state-space planner uses two techniques for making a controller safe:

[2]Where $\mathrm{dwell}(B)$ is the amount of time spent in state B.

- It avoids actions that lead to dangerous states;
- It chooses actions to preempt transitions that lead to dangerous states.

By "dangerous state" we mean, informally, a state that leads to failure, either directly or transitively.

For each reachable state in the plan, CIRCA must compute, for each transition to be preempted, a lower bound on its latency. The latency itself is a property of the path taken to the state. We avoid the explosion in effort by computing bounds for each state that are path *independent*. At every state, we compute a lower bound on the latency, based on the weakest bound from the set of incoming edges.

The latency of a particular temporal transition, t, for a state s, with respect to a predecessor state, p, may be determined according to a limited number of cases:

t is *not* enabled in p: The delay of t does not start "running" until state s is entered. $\mathsf{latency}(t, p \to s) = \mathsf{delay}(t)$.

t is enabled in p: We must count the time spent in state p (and in predecessors to p in which t is enabled); let's refer to this as the dwell in state p. So, recursively:

$$\mathsf{latency}(t, p \to s) = \left(\min_{p' | p' \to p} \mathsf{latency}(t, p' \to p) \right) - \mathrm{dwell}(p)$$

Fortunately, $\mathrm{dwell}(p)$ may easily be determined: the amount of time the system spends in a state is bounded by the action assignment to that state.

Algorithm

We have implemented an algorithm for computing the latencies described above. The bounds we describe above may be computed using a simple depth-first graph search, from nodes to their enabling predecessors. The algorithm has an additional termination condition: terminate when the latency goes to zero. This termination condition allows this algorithm to complete even in plans (graphs) with cycles.

The latency computations are used as an oracle by the planner. When the state-space planner adds edges to the graph (by assigning actions), removes them (by preemption), or, in DAP planning, refines a state description, this algorithm is run to recompute transition latencies. Latency computations may trigger backtracking for two reasons:

- they indicate that the current action assignment fails to achieve necessary preemptions;
- they indicate that, because of a change in the NFA topology, a previous action assignment no longer achieves its preemptions.

Incompleteness

There are classes of real-time plans that CIRCA is unable to find because of the temporal reasoning it does.

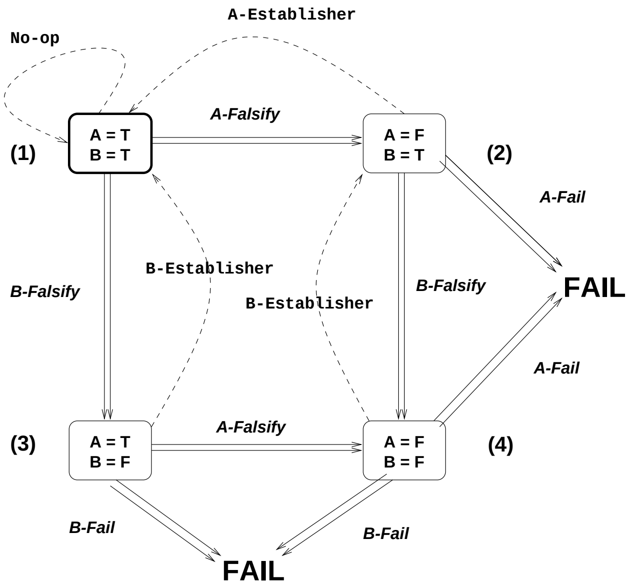


Figure 4: A simple problem unsolvable by the CIRCA planner.

The problem arises because, instead of individually considering the possible ways that the CIRCA NFA could reach a particular state, CIRCA simply computes worst-case bounds on how much time it has to preempt a particular transition. In this section we give a simple example of such a problem.

Figure 4 shows a simple, valid plan that CIRCA is unable to generate.[3] In this situation, there are two bad processes active, *A-Fail* and *B-Fail*. These temporal transitions to failure are active when the corresponding feature has the value false. There are temporal transitions that falsify the corresponding propositions, *A-Falsify* and *B-Falsify*. The CIRCA agent has at its disposal the two actions `A-Establisher` and `B-Establisher`, which make the corresponding propositions true.

Why can't CIRCA find this simple, sound plan? The reason lies in the way the CIRCA planners (both original and DAP) compute temporal bounds in order to determine whether transitions will be preempted. When CIRCA considers whether a state is safe, it computes bounds on the latency of all temporals, based on all the possible ways of reaching that state. These bounds consider the best and worst cases for each transition *independently*. However, in order to determine that some plans, like the one in this example, are safe, one has to consider the interaction between temporals.

Note that this is a problem of the CIRCA planner, not of the CIRCA execution model. The plan given in Figure 4 *is* executable by the RTS. It is not necessary to consider how CIRCA reaches a state in order for it to execute the plan — only in order to determine that

[3] This example is an abstraction of a problem in the Puma robot arm domain.

it can be executed correctly.

Related Work

As we mentioned earlier, Kabanza *et al.* (Kabanza, Barbeau, & St.-Denis 1997) have developed a planning method for reactive agents based on a model similar to ours. Their architecture differs in emphasis, however. The NFAs it constructs are "clocked:" they make transitions at times that are the least common denominator of all possible transitions. This scheme will suffer a state space explosion in domains where there is a wide range of possible transition delays, like those to which CIRCA has been applied.

Dynamic Abstraction Planning

The original CIRCA planner used a forward planning algorithm with full state descriptions. This led to an explosion in the state space of the planner. In recent work (Goldman *et al.* 1997), we have addressed this state-space explosion using a new planning technique that we call Dynamic Abstraction Planning (DAP). Abstraction is used to omit detail from the state representation, reducing both the size of the state space that must be explored to produce a plan, and the size of the resulting plan itself. The abstraction method we describe has three useful features:

1. The abstraction method does not compromise safety-preserving guarantees: the world model used for planning is reduced, but not in ways that affect the system's ability to make rigorous statements about the safety assurances of plans it is building.
2. The method is fully automatic, and dynamically determines the appropriate level of abstraction during the planning process itself.
3. The method uses different levels of abstraction in different parts of the search space, individually adjusting how much detail is omitted at each step.

The intuition behind DAP is fairly simple: in some situations, certain world features are important, while in other situations those same features are not important. An optimal state space representation would capture only the important features for any particular state. In essence, DAP allows a planner to search for useful state space abstractions at the same time it is searching for a plan.

DAP Technique

Recall that the problem of planning for CIRCA (setting aside the question of TAP generation and scheduling), is to assign to every reachable state in the state space, an action that preserves safety. As mentioned above, the original CIRCA planner assigned these actions working forward from a set of start states. The CIRCA planner would maintain a frontier of reachable states and would assign a suitable action to each reachable state.

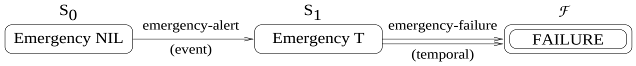


Figure 5: A partially-completed CIRCA plan.

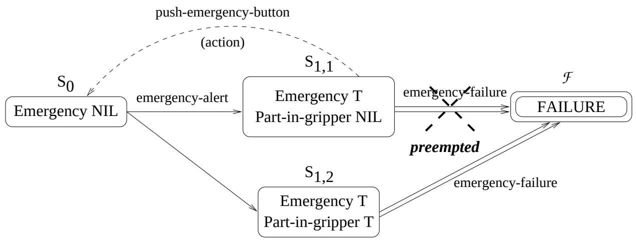


Figure 6: A refinement of the NFA in Figure 5.

In contrast, the DAP planner begins with a very coarse NFA/plan, with all the non-failure states consolidated into a single state. DAP dynamically adds more detail to this sketchy NFA. DAP refines the NFA when it is unable to generate a satisfactory plan at the current level of detail. DAP refines the NFA by taking an existing state and splitting it into a number of more specific states, one for each possible value of a particular feature, F_i.

For example, let us consider the partially-completed plan given in Figure 5. Here there are three states: the failure state and two non-failure states, one for each value of `emergency`, a boolean proposition. This example is based on the domain model given in Figure 2. We assume that `emergency` is `nil` when the system begins operation.

The NFA in Figure 5 is not safe, because there is a reachable state, S_1, from which there is a transition to the failure state (`emergency-failure`) that has not been preempted. One way to fix this problem would be to choose an action for S_1 that will preempt `emergency-failure`. The domain description contains such an action, `push-emergency-button`. Unfortunately, one of `push-emergency-button`'s preconditions is `part-in-gripper= nil` and S_1 is not sufficiently detailed to specify values for `part-in-gripper`. We can rectify this omission by splitting S_1 into a set of states, one for each value of `part-in-gripper`. The resulting NFA is given in Figure 6. We can now assign `push-emergency-button` to solve the problem posed by state $S_{1,1}$. Further planning is required to resolve the problem posed by $S_{1,2}$, either by finding a preempting action that does not require `part-in-gripper` = `nil` or by making $S_{1,2}$ unreachable.

One unusual aspect of DAP is that detail is added to the NFA only *locally*. In our example above, we only added the feature `part-in-gripper` to the part of the state space where the `emergency` feature took on the value `true`, rather than refining all of the states of the NFA symmetrically. This introduces new nondeterminism: because we do not have a complete model of the initial state, we cannot say whether the `emergency-alert` transition will send the system to state $S_{1,1}$ or $S_{1,2}$.

Experimental Evaluation

Dynamic abstraction provides the greatest benefits in domains where uncertainty is present and can be reasoned about or managed in abstract ways, rather than requiring fully-detailed reasoning at all points in the state-space. We have experimented with DAP in randomly-generated domains with various kinds of uncertainty, demonstrating significant speed-ups.

In the CIRCA planning paradigm, there are several possible sources of uncertainty that can be advantageous for DAP:

Events and Temporals — These nonvolitional transitions are outside of the system's control, and therefore lead to uncertainty in the planned trajectory through the state-space, as they can move the system off the direct planned path. The original planner must consider each state that results from an event or temporal and the resulting sequences, possibly not overlapping with the direct plan, that are required to move towards the goal.

Initial conditions — The system can be told it may begin in one of several possible initial conditions. This uncertainty requires the original planner to consider explicit paths leading from each initial state, while DAP may be able to ignore the differences between initial states and find a single plan.

Nondeterministic actions — Because actions can have nondeterministic outcomes, they can cause branching in the original planner that DAP may be able to avoid.

Dynamic Abstraction Planning is not applicable to CIRCA planning alone. The DAP technique could bring automated abstraction to other planners with different state representation, transition semantics, and temporal models. Therefore, we have evaluated the benefits of DAP independent of many of the CIRCA-specific details of the planning model. In particular, to evaluate DAP independent of the complex CIRCA temporal model, we avoid using temporal transitions. In turn, this means that the issue of preemption does not arise in the evaluation problems discussed here.

Our evaluation consisted of running both the DAP and Classic CIRCA state-space planners on numerous domains that were automatically generated to meet several sets of defining characteristics. Each of these sets of domains (or "domain classes") highlights particular ways in which DAP differs from, and usually improves upon, Classic CIRCA and other state-space planners.

In particular, we present results for test domains that focus on:

- "Benign" events;
- Uncertainty in initial conditions;
- Interactions between events and goal achievement.

Our random problem design is based on techniques developed by Barrett and Weld for "classical" planning (Barrett & Weld 1994). Throughout our experiments, we consider domains in which there is a *causal chain* from the initial state(s) to the goal. To automatically generate one of these domains, we build a sequence of simple actions that rely on each others preconditions, and must be chained together to achieve the final goal. Using a notation derived from that used by Barrett and Weld (Barrett & Weld 1994), we can describe these actions by the template:

```
(make-instance 'action :name Achieve-Goal-i
               :preconds ( (G_i F) (G_{i-1} T) )
               :postconds ( (G_i T) ))
```

A set of actions like this creates a sequence of goal features, $G_1, G_2, \ldots G_n$. In our experiments we vary the length of these causal chains and show how this parameter affects run-time and the size of the plan graph generated.

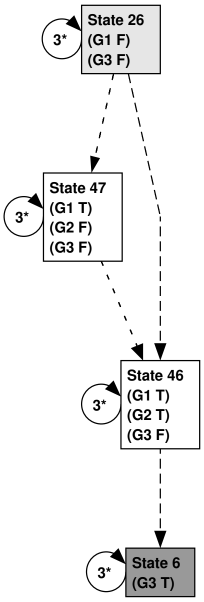
Eval-1 Domain: Benign Events The first evaluation domain, Eval-1, shows DAP's ability to ignore irrelevant nonvolitional transitions. By ignoring these irrelevant transitions, and the features they affect, DAP avoids an exponential state-space explosion that plagued the original state-space planner. In Eval-1 we focus on one type of irrelevant transition, a *benign event*. A benign event is an event establishing a proposition that does not appear in any preconditions or postconditions of actions on the "causal chain" to the goal. The Eval-1 domains show how DAP can build small, abstract plans that accurately characterize much larger state spaces connected by benign events. On the other hand, the original planner must enumerate the entire exponential state spaces.

To introduce benign events, we create an additional set of "external" features that are irrelevant to the causal chain. The values of these external features may change over time due to events. We define a number of events that establish these propositions. These events do not interact with the causal chain, for good or ill.

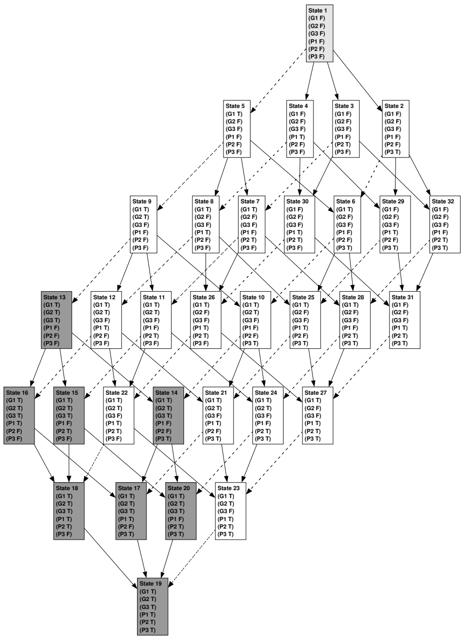
Figure 7 shows the different structure of the plans generated by DAP and the classic CIRCA planner. These pictures illustrate the results for one Eval-1 domain, in which the causal chain is of length three, there are three external features and three establisher events.

The performance of the two planners on this highly-structured domain can be predicted analytically. As the number of benign events increases, the DAP planner's final plan size does not change at all. All the events are sufficiently modeled by the simple self-loops shown in Figure 7(a). On the other hand, the original CIRCA planner's state space grows exponentially. Each benign event in the domain forces the original planner to replicate the entire path from initial state to goal (which is $n+1$ states long). For each of the 2^m combinations of values of the external features, there is one such path. So the size of the original planner's state space is $(n+1) * 2^m$. We have confirmed this relationship experimentally.

Figure 8 shows that the savings in state-space translate to savings in terms of runtime. As expected, DAP runtime is linear in both the number of goals and be-



(a) DAP plan.



(b) Classic CIRCA plan.

Figure 7: Plans for the Eval-1 domain with 3 goals and 3 benign events. In the DAP plan, the 3* notation indicates that the self-loops are triply-replicated, corresponding to the events affecting the three external events not included in each abstract state. In all diagrams, initial states are lightly shaded, goal states are darker.

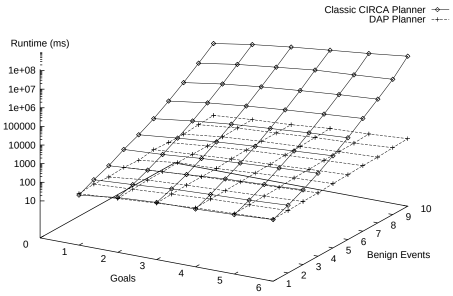


Figure 8: Classic CIRCA's runtime is exponential in the number of benign events. Note the logarithmic runtime scale.

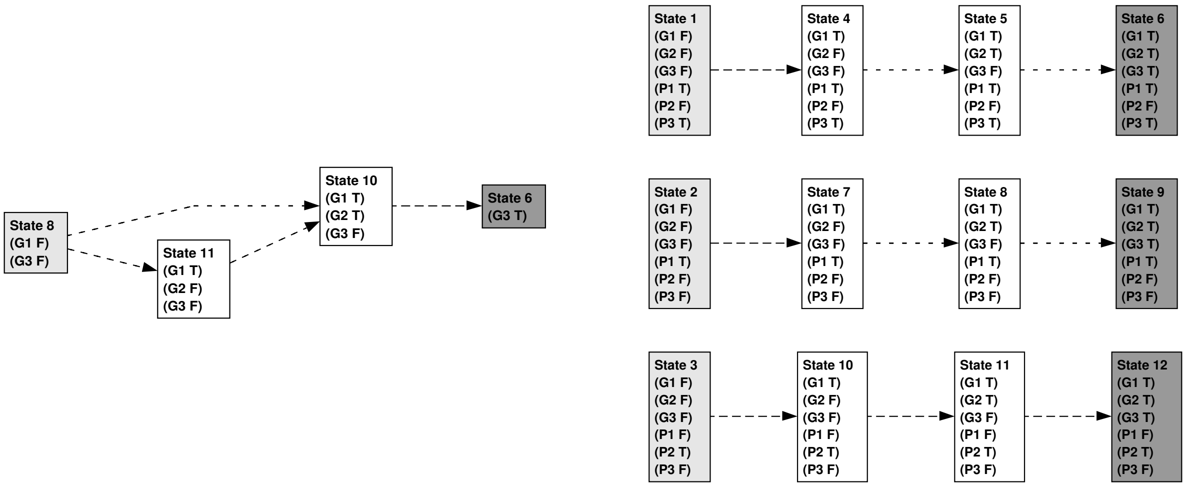
nign events, while the old state-space planner's runtime grows exponentially in the number of benign events.

Eval-2 Domain: Uncertainty in Initial Conditions This domain class shows the advantages of DAP when there is uncertainty in the initial conditions of the planning problem. Unlike Eval-1, there are no external events at all; instead, the only uncertainty arises in the initial conditions. Multiple initial states are created by adding external features and randomly choosing value assignments to them. The randomness serves only to build different initial conditions.

On Eval-2 domains, DAP completely ignores the differences between the declared initial conditions, building only a single abstract start state. This yields substantial savings in the number of states enumerated and in the planner's runtime. For example, Figure 9a shows the DAP plan for an Eval-2 domain declared with three goal predicates and three initial states. The graph shows only one initial state because DAP never splits the state space on any of the predicates that differentiate the declared initial states. In contrast, Figure 9b shows the Classic planner's output, in which each of the initial states leads to a different path through the state-space. As this example suggests, for Eval-2 domains DAP's plan size is constant with respect to the number of initial conditions, while the Classic plans grow linearly (see Figure 10).

Eval-3 Domain: Required Events The Eval-3 domain class investigates planner performance when the planner must incorporate nonvolitional transitions (events) into the path to the goal. To force the planner to rely on events in the planned path, we made each goal-achieving action include a single external predicate as a precondition. These external propositions can only be established by events; there are no actions that establish them. Complicating matters further, we specified new events that can make the external predicates false, or "delete" them, in addition to the original "adding" events from the Eval-1 domain.

Ideally, DAP would perform a single split on one ex-



(a) DAP. (b) Classic.

Figure 9: Plans for Eval-2 domain with 3 goals and 3 initial states.

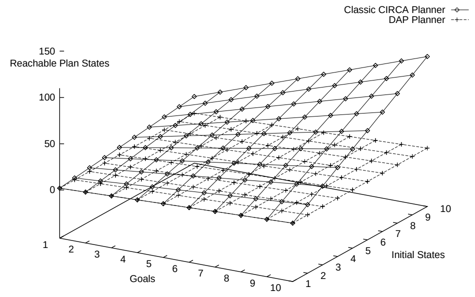


Figure 10: Plan size for Eval-2 domains with uncertainty in initial conditions.

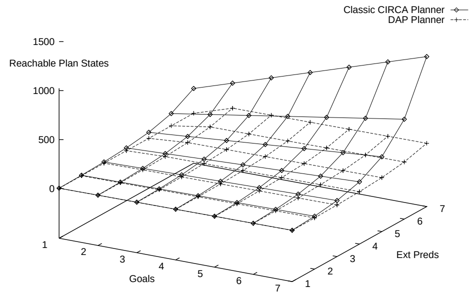


Figure 11: Plan size for Eval-3 domains with required events.

ternal predicate and then rely on that predicate and the correspondingly-enabled actions for the rest of the plan. This would make DAP's plans only slightly larger than for the Eval-1 domains. Unfortunately, in the course of doing this experiment, we discovered that the heuristic DAP uses to choose state refinements is not smart enough to recognize that it is enough to know *one* external predicate. In fact, the current heuristic code often does as poorly as possible! Figure 11 shows that even with this poor heuristic, DAP achieves a substantial savings in state-space size. Figure 12 shows that the state-space savings is not enough to substantially improve runtime over the original planner, because of the bookkeeping costs imposed by DAP. Since the heuristic code here is misleading DAP badly, this is very much a worst case. We have identified the flaw in the heuristic and a fix for it; we will repair it in future work.

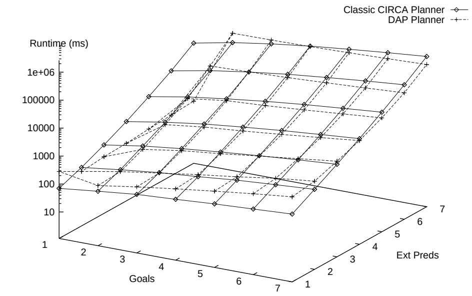
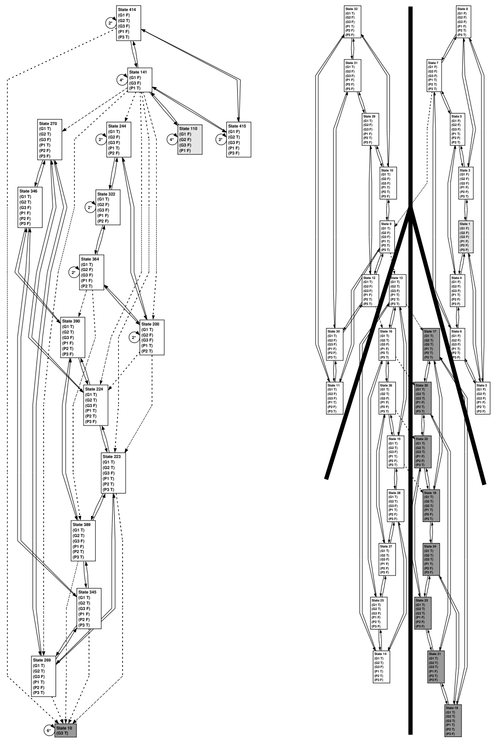


Figure 12: Runtime for Eval-3 domains with required events.



(a) DAP. (b) Classic.

Figure 13: Plans for Eval-4 domain with 3 goals.

Eval-4 Domain: Complex Event Interleaving By modifying the Eval-3 domain class slightly, we produced a new class designed to highlight DAP's ability to abstract the state space in a non-homogeneous fashion, including a feature in some parts of the space, and ignoring it in others. Eval-4 domains consist of a set of goal-achieving actions that each require a *different* external predicate.

Thus an n-goal Eval-4 domain also has n events in the success path, and at some point even the DAP planner will have to consider all n external predicates. However, DAP can limit the propagation of those external predicates so that it still does not consider the exponential set of their combinations, as illustrated by the example in Figure 13a. The Classic CIRCA planner, on the other hand, must enumerate all combinations , and builds plans like that shown in Figure 13b. The overall performance results are shown in Figure 14.

Comparison to Other State-space Abstraction Techniques

Many classical planning systems have used abstraction methods to increase the efficiency of searching for plans (see (Kambhampati 1994) for a brief survey). However, these abstractions are typically used only as guides in searching for a plan; the system may not know that its goals will actually be achieved by an abstract plan, and it will not be able to execute the abstracted operators directly. Instead, traditional abstraction planners must eventually expand their current plans down to the lowest level of detail, removing the abstraction to produce a final executable plan.

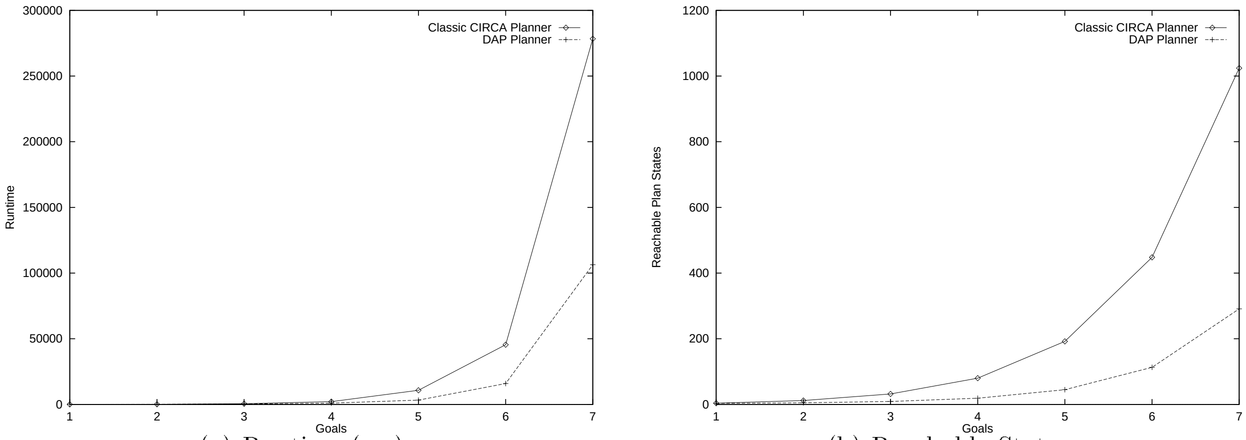
In the DAP approach, which involves abstraction only of state descriptions, abstract plans are executable, because the operators are always completely specified. This has two main advantages. First, the planning process can supply initial plans that preserve safety but might, on further refinement, do a better job of goal achievement. Second, the planning process can terminate with an executable abstract plan, which our results have shown may be much smaller than the corresponding plan expanded to precisely-defined states.

Dearden and Boutilier (1997) have developed an abstract planning algorithm for decision-theoretic planning modeled as a Markov decision process (MDP). Their method is similar to the DAP approach in that it involves aggregating states, but there are some differences. First, their method is not dynamic: aggregation is performed using a predefined set of "relevant" propositions, which is determined using Knoblock's approach (Knoblock 1994). Second, their method is uniform: the same propositions are relevant everywhere. The underlying model is also significantly different from CIRCA's: it does not model exogenous events or the timing required for real-time guarantees.

In more recent work, Boutilier *et al.* have developed an approach to MDPs that uses dynamic, local abstraction, much like our own (Boutilier, Dearden, & Goldszmidt 1995). Their technique is like ours in gradually, dynamically adding information to different parts of the state space, and in using regression across actions to direct state refinement. The technique differs substantially because of the differences between CIRCA and MDP planning models: CIRCA has a very detailed temporal model and multiple, asynchronous environmental processes, but a very weak model of uncertainty. On the other hand, MDPs have a weak temporal model (effectively, all processes are "clocked" at the same rate), but a very sophisticated model of uncertainty.

Our DAP planning algorithm is essentially performing on-line model minimization. This model minimization is based on concepts of regression from "classical" AI planning. In work done simultaneously with our development of DAP, Givan and Dean have explored the connection between model minimization and STRIPS-style planning (Givan & Dean 1997). They show how STRIPS style regression may be interpreted in terms of model minimization. While our work is similar to theirs in exploiting classical planning-inspired notions of regression, our work is closer to mainstream work in model minimization, since it is focused on NFAs, rather than on finding a single path to the goal.

Godefroid and Kabanza (1991) have developed an abstraction technique based on partial orders. Their results allow a system to examine only a single ordering of independent actions, rather than enumerating



(a) Runtime (ms). (b) Reachable States.

Figure 14: Eval-4 domain shows DAP using non-homogeneous abstraction to advantage.

all possible orderings. Unfortunately, these results are not immediately applicable to CIRCA, because their world model does not include exogenous events. The more recent work by Kabanza *et al.* (Kabanza, Barbeau, & St.-Denis 1997) *does* include exogenous events, but they do not seem to have carried over the earlier abstraction concepts.

Conclusions

We have presented two abstraction techniques used in CIRCA's automatic generation of hard real-time discrete event controllers. The controller synthesis is handled by constructing a timed NFA. In order to avoid state-space explosion, we abstract both the temporal and feature space representation. The time space is abstracted by using a bounding calculus to avoid having to explicitly use a non-Markov representation. Approaches that reason explicitly about paths would make it impossible for us to plan in the domains of interest to us. The feature space is compressed through dynamic abstraction. We have shown that second abstraction method provides a substantial savings in domains that feature uncertainty.

References

Barrett, A., and Weld, D. 1994. Partial order planning: Evaluating possible efficiency gains. *Artificial Intelligence* 67(1):71–112.

Boutilier, C.; Dearden, R.; and Goldszmidt, M. 1995. Exploiting structure in policy construction. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, 1104–1113.

Dearden, R., and Boutilier, C. 1997. Abstraction and approximate decision-theoretic planning. *Artificial Intelligence* 89(1–2):219–283.

Givan, R., and Dean, T. 1997. Model minimization, regression and propositional STRIPS planning. In Pollack, M., ed., *Proceedings of the 15th International Joint Conference on Artificial Intelligence*. Morgan Kaufmann Publishers, Inc.

Godefroid, P., and Kabanza, F. 1991. An efficient reactive planner for synthesizing reactive plans. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, 640–645. Cambridge, MA: MIT Press.

Goldman, R. P.; Musliner, D. J.; Krebsbach, K. D.; and Boddy, M. S. 1997. Dynamic abstraction planning. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, 680–686. Menlo Park, CA: American Association for Artificial Intelligence.

Kabanza, F.; Barbeau, M.; and St.-Denis, R. 1997. Planning control rules for reactive agents. *Artificial Intelligence* 95(1):67–113.

Kambhampati, S. 1994. Refinement search as a unifying framework for analyzing planning algorithms. In Doyle, J.; Sandewall, E.; and Torasso, P., eds., *Principles of Knowledge Representation and Reasoning:Proceedings of the Fourth International Conference*. Morgan Kaufmann Publishers, Inc.

Knoblock, C. A. 1994. Automatically generating abstractions for planning. *Artificial Intelligence* 68:243–302.

Musliner, D. J.; Durfee, E. H.; and Shin, K. G. 1993. CIRCA: a cooperative intelligent real-time control architecture. *IEEE Transactions on Systems, Man and Cybernetics* 23(6):1561–1574.

Musliner, D. J.; Durfee, E. H.; and Shin, K. G. 1995. World modeling for the dynamic construction of real-time control plans. *Artificial Intelligence* 74(1):83–127.

Ostroff, J. S., and Wonham, W. M. 1990. A framework for real-time discrete event control. *IEEE Transactions on Automatic Control* 35(4):386–397.